Doom 3 → Dante

Performance on Mesa (i965)
(Not a Demo!)

Quick overview of Doom 3

- GPLv3+ (with additional terms) on November 22, 2011.
  - Without “Carmack's Reverse” (aka depth fail) shadows.

    ```
    Date: Fri, 25 Nov 2011 01:32:56 +0200

    Subject: [PATCH v3 1/1] renderer: added support for Carmack's Reverse (depth
    fail) shadows.
    ```
- OpenGL 1.x + OpenGL extensions.
  - Some of which are requirements.
- X11 and GLX.
- 8 years old.
- ARB2 backend (best backend)
  - ARB_vertex_program && ARB_fragment_program.
  - Other backends available for older hardware.
- ARB_vertex_buffer_object used when available, otherwise fallback to virtual memory.

Quick overview of Dante

- OpenGL ES2.0
  - EGL
  - GLSL primary backend
    - ARB2 backend remains for debugging on the desktop; stubbed out when compiled for ES2.0
    - Carmack's Reverse (depth fail) added back
  - VBO requirement
  - ARBvp and ARBfp programs are *not* part of the GPLv3+ release
    - “Clean-room” programs written in GLSL
    - Phong (rather than Blinn-Phong) shading model.
      - More computationally expensive but delivers much more realistic rendering.
    - Optional Half-Lambert lighting (see example on next slide: Phong + Half-Lambert.)
- Support for Android...
  - You'd better have a high-end device!
  - “Some” bugs and missing features...

Lambert vs Half-Lambert

Half-Lambert Gone Wrong?

Optimization on Mesa

- Unfortunately no really great tools for Mesa performance analysis...
  - i965: intel_gpu_top: works like regular `top' –
    - No support for per-frame analysis,
    - No support for pretty graphs (unless you're into ASCII art.)
  - Useful for rough estimate of GPU load.
  - Basically unusable output for game devs who haven't read and understood Intel HW docs.
    - Game devs typically don't want to read low-level HW docs...

- So, what should we do to fix this for Mesa drivers?
  - Quick example of intel_gpu_top first...

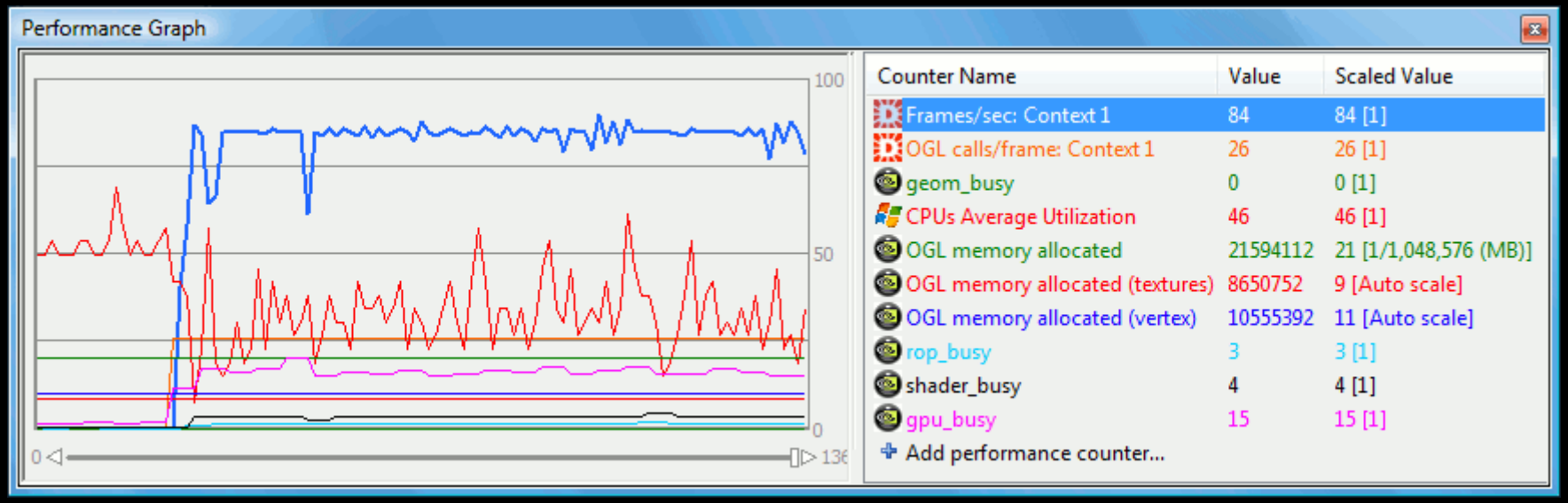
intel_gpu_top

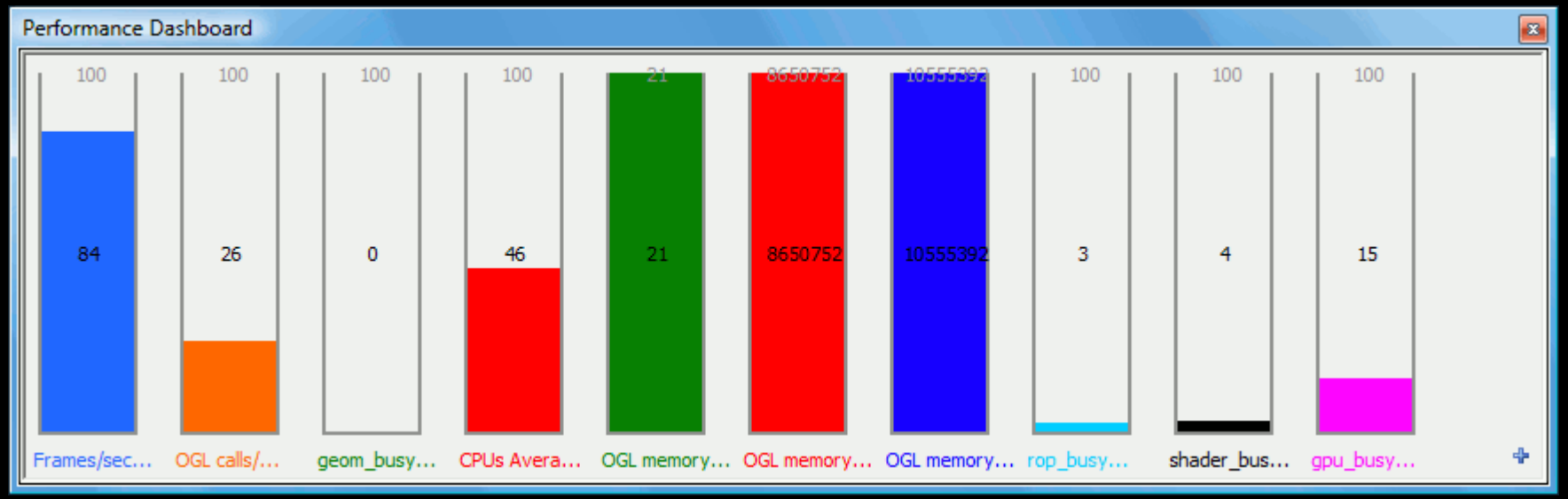
```
   render busy:  37%: ########                render space: 69/131072
bitstream busy:   0%:                     bitstream space: 0/131072
  blitter busy:  36%: ########               blitter space: 30/131072

         task  percent busy
          GAM:  68%: ##############         vert fetch: 0 (0/sec)
           CS:  37%: ########               prim fetch: 0 (0/sec)
          PSD:  32%: #######             VS invocations: 33076780 (1617385/sec)
          DAP:  28%: ######              GS invocations: 0 (0/sec)
        RCPFE:  28%: ######                   GS prims: 0 (0/sec)
           IZ:  28%: ######              CL invocations: 16538390 (808744/sec)
        RCPBE:  28%: ######                   CL prims: 11324693 (689777/sec)
          RCC:  28%: ######              PS invocations: 11415570625 (347597257/sec)
         WMFE:  28%: ######              PS depth pass: 11132520857 (340957947/sec)
        EU 30:  26%: ######
          SVG:  26%: ######
        EU 10:  25%: #####
        EU 00:  25%: #####
          HIZ:  25%: #####
        EU 20:  25%: #####
           TD:  25%: #####
         SVRW:  25%: #####
         IC 3:  23%: #####
         WMBE:  23%: #####
         IC 2:  23%: #####
         IC 0:  23%: #####
         IC 1:  23%: #####
        EU 01:  22%: #####
```

Better debugging/analysis tools!

- AMD's gDEBugger works on GNU/Linux, but only with AMD hardware and fglrx.
  - Older pre-AMD versions used to run with Mesa, but have problems with modern glibc.
  - Proprietary tool (both pre and post-AMD versions.)
  - Basically unusable for me...
- Nvidia, SGX, ... have similar tools for their proprietary drivers.
- We don't have any great tools for Mesa...
  - But we should!


Performance Graph
100
50
0
0
Counter Name
Value
Scaled Value
Frames/sec: Context 1
84
84 [1]
OGL calls/frame: Context 1
26
26 [1]
geom_busy
0
0 [1]
CPUs Average Utilization
46
46 [1]
OGL memory allocated
21594112
21 [1/1,048,576 (MB)]
OGL memory allocated (textures)
8650752
9 [Auto scale]
OGL memory allocated (vertex)
10555392
11 [Auto scale]
rop_busy
3
3 [1]
shader_busy
4
4 [1]
gpu_busy
15
15 [1]
Add performance counter...


Performance Dashboard
100
84
Frames/sec...
100
26
OGL calls/...
100
0
geom_busy...
100
46
CPUs Avera...
21
21
OGL memory...
8650752
8650752
OGL memory...
10555392
10555392
OGL memory...
100
3
rop_busy...
100
4
shader_bus...
100
15
gpu_busy...

Linux kernel and `perf' system...

- **https://perf.wiki.kernel.org**
  - Stumbled across this by accident while looking at CPU profiling.

`perf` provides rich generalized abstractions over hardware specific capabilities. Among others, it provides per task, per CPU and per-workload counters, sampling on top of these and source code event annotation.

  - `perf stat`: obtain event counts
  - `perf record`: record events for later reporting
  - `perf report`: break down events by process, function, etc.
  - `perf annotate`: annotate assembly or source code with event counts
  - `perf top`: see live event count

Kernel `perf' system and Mesa

- Possibly create infrastructure in DRM and hook into `perf' sub-system?
- Needs some cooperation with userspace:
  - Mesa should indicate frame termination without causing a stall, e.g.
    - **OUT_BATCH(SCRATCH_REG_0, 0xDEADD00D);**
  - Could be done at swap buffers or more intelligently with the ***GL_GREMEDY_frame_terminator*** extension (with application support.)
- Userspace debugger could read the data from kernel and generate pretty graphs, suggestions, etc.
  - Interactive GUI,
  - HTML report,
  - ASCII art. **;** -)
- Very much hand waving at this point. No prototype implementation.

Mesa debug output

- Mesa drivers may be able to provide “hints” for the OpenGL application:

```
if (ctx->Scissor.Enabled)
   perf_debug("Failed to fast clear depth due to scissor being enabled.
              Possible 5%% performance win if avoided.\n");
```

  - 20 dwords to change surface state (disable the scissor test.)
  - How to synchronize these with the data from kernel `perf' system?
  - Possibly with a carefully managed frame counter?

- Userspace debugger could match frame counter of data fetched from `perf' system and strings fetched from *ARB_debug_output*.
  - Currently `perf_debug()` ***does not*** output to *ARB_debug_output!*

- *ARB_debug_output* works as long as the debugger and OpenGL application are in the same context...
  - But we probably do not want such a solution; it's ugly and we lose any benefits of having the debugger as a separate process.
  - Not quite sure how to handle Mesa debug output with the debugger in a separate process... Suggestions?

GLX vs EGL

- Dante (OpenGL ES2.0, X11 (XCreateWindow et al), **EGL**):
  - +timedemo demo1
  - **vblank_mode=0**

  2148 frames rendered in 64.6 seconds = 33.3 fps

  MessageBox: Time Demo Results - 2148 frames rendered in 64.6 seconds = ***33.3 fps***

- Dante (OpenGL ES2.0, X11 (XCreateWindow et al), **GLX**):
  - +timedemo demo1
  - **vbank_mode=0**

  2148 frames rendered in 47.2 seconds = 45.5 fps

  MessageBox: Time Demo Results - 2148 frames rendered in 47.2 seconds = ***45.5 fps***

- Mesa appears to ignore *vblank_mode* in the EGL code...

```
src/egl/drivers/dri2/platform_x11.c-   struct dri2_egl_surface *dri2_surf =
dri2_egl_surface(surf);
src/egl/drivers/dri2/platform_x11.c-#endif
src/egl/drivers/dri2/platform_x11.c-
src/egl/drivers/dri2/platform_x11.c:   /* XXX Check vblank_mode here? */
src/egl/drivers/dri2/platform_x11.c-
src/egl/drivers/dri2/platform_x11.c-   if (interval > surf->Config->MaxSwapInterval)
src/egl/drivers/dri2/platform_x11.c-      interval = surf->Config->MaxSwapInterval;
```

Conclusion

- Bottom line: We need better performance analysis tools.
- Intel has done work on Mesa/i965 optimization with Valve Software for their “Left 4 Dead 2” game:
  - Eric Anholt, Ian Romanick, and Ken Graunke at Valve's headquarters in person.
  - Possible for a large game development studio,
  - Not possible for indie game developers.
- Performance tools will never be as good as experts in person, but can still be very useful.

Questions? / Comments?